Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 239 (2009) 73–83

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Branching-Time Model-Checking of Probabilistic Pushdown Automata

Tomáš Brázdil[1] Václav Brožek[2] Vojtěch Forejt[3]

Faculty of Informatics
Masaryk University
Brno, Czech Republic

Abstract

In this paper we study complexity of the model-checking problem for probabilistic pushdown automata (pPDA) and qualitative fragments of two branching-time logics PCTL* and PECTL*. We prove that this problem is in **2-EXPTIME** for pPDA and qualitative PCTL*. Consequently, we prove that model-checking of stateless pPDA (pBPA) and both qualitative PCTL* and qualitative PECTL* is **2-EXPTIME**-hard. These results combined with results of several other papers give us that the model-checking problem for pPDA (and also for pBPA) and both qualitative PCTL* and qualitative PECTL* is **2-EXPTIME**-complete. Finally, we survey known results on model-checking of pPDA and branching-time logics.

Keywords: model-checking, Markov chains, temporal logics, probabilistic pushdown automata

1 Introduction

Model-checking is a popular verification method working as follows: Given a model of a system and a desired property expressed by a formula of a suitable logic, decide whether the system satisfies the formula. In this paper we consider this problem for systems modeled by discrete time Markov chains and properties expressed by formulae of branching-time probabilistic logics.

The model-checking problem has been deeply studied for finite Markov chains (see, e.g., [15,9,2]). Several years ago model-checking of various kinds of infinite Markov chains was taken into consideration. The most prominent classes of infinite Markov chains studied in the context of model-checking are chains generated by probabilistic lossy channel systems (see, e.g., [3,1]), and two expressively equivalent classes of recurrent probabilistic systems: probabilistic pushdown automata [10,6]

[1] Email: xbrazdil@fi.muni.cz
[2] Email: xbrozek@fi.muni.cz
[3] Email: xforejt@fi.muni.cz

doi:10.1016/j.entcs.2009.05.031

and recursive Markov chains [14,12,13]. In this paper we concentrate on model-checking of probabilistic pushdown automata (pPDA), and stateless pPDA (pBPA).

Many temporal logics have been designed for specifying properties of Markov chains. In this paper we concentrate on two branching time temporal logics PCTL* and PECTL*. The logic PCTL* is a probabilistic variant of the well-known logic CTL*. The logic PECTL* generalizes PCTL* by substituting path subformulae with non-deterministic Büchi automata (it can also be seen as a probabilistic version of the logic ECTL* defined, e.g., in [8]). To give a complete picture of branching-time model-checking of pPDA, we also consider the logic PCTL which is a probabilistic variant of CTL.

The main aim of this paper is to present complete complexity results on model-checking pPDA and qualitative fragments of PCTL, PCTL* and PECTL* (denoted qPCTL, qPCTL* and qPECTL*, resp.). This paper can be seen as a culmination of more than two years of intensive research (see, e.g., [10,6,14,12,13]). The first paper considering model-checking of pPDA was [10] where (among other results) decidability of model-checking of pPDA and qPCTL was proved. The paper [10] also introduced crucial tools for dealing with model-checking of pPDA and showed that properties expressible using deterministic Büchi automata can effectively be verified for pPDA. The paper [10] was followed by [6] where the model-checking problem for pPDA and logics qPCTL, qPCTL* and qPECTL* was shown to be in **EXPSPACE**, **3-EXPSPACE**, and **2-EXPSPACE**, respectively. Moreover, the paper [6] proved undecidability of the model-checking problem for pPDA and (quantitative) PCTL (this problem still remains open for pBPA and PCTL), and also gave an **EXPTIME**-lower bound on this problem for pBPA and qPCTL (the technique of this proof is based on unpublished observations of Richard Mayr). Consequently, results of Etessami and Yannakakis [12,13] on model-checking linear time properties of pPDA gave rise to an improvement of the upper bounds for qPCTL, qPCTL* and qPECTL* to **EXPTIME**, **3-EXPTIME**, and **2-EXPTIME**, respectively, presented in [5]. This showed that the model-checking problem for pPDA as well as pBPA and qPCTL is **EXPTIME**-complete. Matching bounds for qPCTL* and qPECTL* have not been established so far.

Detailed complexity estimates presented in [5] reveal that the *program* complexity (i.e., the complexity in the size of pPDA when the formula is fixed) of model-checking pPDA (or pBPA) and each of the three logics above is in **EXPTIME** (or in **P**, resp.). On the other hand, in [4] Bozzelli proved that program complexity of model-checking non-probabilistic PDA and CTL is **EXPTIME**-hard. The proof of Bozzelli can easily be transformed to deal with pPDA and qPCTL by assigning arbitrary nonzero probabilities to transitions and by substituting path quantifiers A and E of CTL with $\mathcal{P}^{=1}$ and $\mathcal{P}^{>0}$, respectively. Hence, the program complexity of the model-checking problem for pPDA and each of the logics qPCTL, qPCTL* and qPECTL* is **EXPTIME**-complete. Finally, the **P**-hardness of the program complexity of the model-checking problem for pBPA follows immediately from the **P**-hardness of the emptiness problem for context-free grammars.

In this paper we complete the picture by proving the following results.

Table 1
branching-time model-checking of pPDA (combined)

	qPCTL	qPECTL*	qPCTL*
pBPA	**EXPTIME**-complete	**2-EXPTIME**-complete	**2-EXPTIME**-complete
pPDA	**EXPTIME**-complete	**2-EXPTIME**-complete	**2-EXPTIME**-complete

Table 2
program complexity of branching-time model-checking of pPDA

	qPCTL	qPECTL*	qPCTL*
pBPA	**P**-complete	**P**-complete	**P**-complete
pPDA	**EXPTIME**-complete	**EXPTIME**-complete	**EXPTIME**-complete

- The model-checking problem for pPDA and qPCTL* is in **2-EXPTIME**. To prove this we extend techniques of [13] to show that the set of configurations satisfying a given qPCTL* formula is accepted by a finite state automaton computable in doubly exponential time.
- Model-checking of pBPA and both qPCTL* and qPECTL* is **2-EXPTIME**-hard. In the case of qPCTL* the proof combines techniques of [6] together with the technique used in [4] for showing **2-EXPTIME**-hardness of model-checking non-probabilistic PDA and CTL*. The proof for qPECTL* introduces new observations needed to deal with qPECTL* formulae.

The above results complete the picture of complexity of the model-checking problem for pPDA as well as pBPA and the three logics qPCTL, qPECTL* and qPCTL*. These results are summarized in Table 1 (combined complexity) and Table 2 (program complexity).

2 Preliminaries

A (discrete) *Markov chain* is a triple $M = (S, \to, Prob)$ where S is a finite or countably infinite set of *states*, $\to \subseteq S \times S$ is a *transition relation*, and $Prob$ is a function which to each transition $s \to t$ of M assigns its probability $Prob(s \to t) \in (0, 1]$ so that for every $s \in S$ we have $\sum_{s \to t} Prob(s \to t) = 1$.

We also write $s \xrightarrow{x} t$ instead of $Prob(s \to t) = x$. A *path* in M is a finite or infinite sequence $w = s_0, s_1, \ldots$ of states such that $s_i \to s_{i+1}$ for every i. We denote w_i the path $s_i, s_{i+1}, \ldots$. We use $w(i)$ to denote the state s_i of w. A state t is *reachable* from a state s if there is a finite path starting in s and ending in t. A *run* is an infinite path. The set of all runs is denoted Run.

The sets of all paths, finite paths and runs that start with a given finite path w are denoted $Path(w)$, $FPath(w)$ and $Run(w)$, respectively. In particular, $Run(s)$, where $s \in S$, is the set of all runs initiated in s.

We are interested in probabilities of certain events associated with runs. To every $s \in S$ we associate the probability space $(Run(s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all *basic cylinders* $Run(w)$ where $w \in FPath(s)$, and $\mathcal{P} : \mathcal{F} \to [0, 1]$ is the unique probability measure (see [17,16]) such that $\mathcal{P}(Run(w)) = \Pi_{i=1}^{m} x_i$ where

$w = s_0, \ldots, s_m$ and $s_{i-1} \xrightarrow{x_i} s_i$ for every $1 \leq i \leq m$ (we put $\mathcal{P}(Run(s)) = 1$).

2.1 Branching-Time Temporal Logics

We start with a definition of the temporal logic PCTL*. The syntax of PCTL* *state* and *path* formulae Φ and φ, resp., is given by the following abstract syntax equations.

$$\Phi ::= a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}^{\sim\varrho}\varphi$$
$$\varphi ::= \Phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

Here a ranges over a countably infinite set Ap of atomic propositions, $\varrho \in [0,1]$, and $\sim \in \{\leq, <, \geq, >, =\}$.

Standard abbreviations $\mathtt{tt}$, $\vee$ and $\Rightarrow$ from the propositional logic are used, moreover $\Diamond\varphi \equiv \mathtt{tt}\ \mathcal{U}\, \varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. The logic PCTL is a fragment of PCTL* where path formulae are given by the equation $\varphi ::= \mathcal{X}\Phi \mid \Phi_1 \,\mathcal{U}\, \Phi_2$.

In order to define the logic PECTL* we need the notion of a Büchi automaton. A Büchi automaton is a tuple $\mathcal{B} = (B, \Sigma, \rightarrow, q_I, F)$, where Σ is a finite *alphabet*, B is a finite set of *states*, $\rightarrow \subseteq B \times \Sigma \times B$ is a *transition relation* (we write $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \rightarrow$), q_I is the *initial state*, and $F \subseteq B$ is a set of accepting states. The automaton $\mathcal{B}$ accepts a word $w \in \Sigma^\omega$ if there exists a sequence $q_0, q_1, \ldots$ of states of $\mathcal{B}$ such that $q_0 = q_I$, for all $i \geq 0$ we have $q_i \xrightarrow{w(i)} q_{i+1}$, and some state of F occurs infinitely many times in $q_0, q_1, \ldots$. We denote $\mathcal{L}(\mathcal{B})$ the set of all words accepted by $\mathcal{B}$. The size of the automaton $\mathcal{B}$ is defined to be $|\rightarrow|$.

The logic PECTL* has only state formulae with the following syntax:

$$\Phi ::= a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}^{\sim\varrho}\mathcal{B}(\Phi_1, \ldots, \Phi_n)$$

Here $n \geq 1$, $\mathcal{B}$ is a Büchi automaton over an alphabet $\Sigma \subseteq 2^{\{1,\ldots,n\}}$, and each Φ_i is a PECTL* formula.

The semantics of PCTL* formulae is defined as follows. Let $M = (S, \rightarrow, Prob)$ be a Markov chain and let $\nu : Ap \rightarrow 2^S$ be a valuation. State formulae are interpreted over S, and path formulae are interpreted over Run. Hence, for a given $s \in S$ and $w \in Run$ we define

$s \models^\nu a$	iff	$s \in \nu(a)$
$s \models^\nu \neg\Phi$	iff	$s \not\models^\nu \Phi$
$s \models^\nu \Phi_1 \wedge \Phi_2$	iff	$s \models^\nu \Phi_1$ and $s \models^\nu \Phi_2$
$s \models^\nu \mathcal{P}^{\sim\varrho}\varphi$	iff	$\mathcal{P}(\{w \in Run(s) \mid w \models^\nu \varphi\}) \sim \varrho$

$w \models^\nu \Phi$	iff	$w(0) \models^\nu \Phi$
$w \models^\nu \neg\varphi$	iff	$w \not\models^\nu \varphi$
$w \models^\nu \varphi_1 \wedge \varphi_2$	iff	$w \models^\nu \varphi_1$ and $w \models^\nu \varphi_2$
$w \models^\nu \mathcal{X}\varphi$	iff	$w_1 \models^\nu \varphi$
$w \models^\nu \varphi_1 \,\mathcal{U}\, \varphi_2$	iff	$\exists j \geq 0 : w_j \models^\nu \varphi_2$ and $w_i \models^\nu \varphi_1$ for all $0 \leq i < j$

The semantics of a PECTL* formula $\Phi = \mathcal{P}^{\sim\varrho}\mathcal{B}(\Phi_1, \ldots, \Phi_n)$ is defined as follows: First, we can assume that the semantics of the PECTL* formulae $\Phi_1, \ldots, \Phi_n$ has already been defined. This means that for each $w \in Run$ we can define an infinite word w_Φ over the alphabet $2^{\{1,\ldots,n\}}$ by $w_\Phi(i) = \{k \in \{1, \ldots, n\} \mid w(i) \models^\nu \Phi_k\}$ for all $i \geq 0$. For every state s, let $Run(s, \Phi) = \{w \in Run(s) \mid w_\Phi \in \mathcal{L}(\mathcal{B})\}$. We stipulate that $s \models^\nu \Phi$ if and only if $\mathcal{P}(Run(s, \Phi)) \sim \varrho$.

The logic PECTL* is more expressive than PCTL*. Indeed, using standard algorithms for conversion of LTL formulae to Büchi automata we can assign to every PCTL* state formula Φ a semantically equivalent PECTL* formula of size

exponential in $|\Phi|$.

The *qualitative fragments* of PCTL, PCTL*, and PECTL*, denoted qPCTL, qPCTL*, and qPECTL*, resp., are obtained by restricting the allowed operator/number combinations in $\mathcal{P}^{\sim\varrho}\varphi$ and $\mathcal{P}^{\sim\varrho}\mathcal{B}(\Phi_1,\ldots,\Phi_n)$ subformulae to '=0' and '=1'. (Observe that '<1', '>0' are definable from '=0', '=1', and negation.)

The model-checking problem for the above logics is stated as follows.

Given a state s_0 of M, a valuation ν, and a *state* formula Φ, does $s_0 \models^\nu \Phi$ hold?

2.2 Probabilistic PDA

A *probabilistic PDA (pPDA)* is a tuple $\Delta = (Q, \Gamma, \delta, Prob)$ where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma^{\leq 2}$ (where $\Gamma^{\leq 2} = \{\alpha \in \Gamma^*, |\alpha| \leq 2\}$) is a *transition relation*, and $Prob$ is a function which to each transition $pX \to q\alpha$ assigns a rational probability $Prob(pX \to q\alpha) \in (0,1]$ so that for all $p \in Q$ and $X \in \Gamma$ we have that $\sum_{pX \to q\alpha} Prob(pX \to q\alpha) = 1$.

In the rest of this paper we write $pX \to q\alpha$ instead of $(p, X, q, \alpha) \in \delta$, and $pX \xrightarrow{x} q\alpha$ instead of $Prob(pX \to q\alpha) = x$. We also assume that for each $pX \in Q \times \Gamma$ there is at least one $q\alpha$ such that $pX \to q\alpha$. The set $Q \times \Gamma^*$ of all configurations of Δ is denoted by $\mathcal{C}(\Delta)$. Given a configuration $pX\alpha$, we call pX its *head* (a head of $p\varepsilon$ is p).

We denote pBPA the class of all pPDA with just one control state. In what follows, configurations of pBPA are usually written without the control state (i.e., we write only α instead of $p\alpha$).

To Δ we associate the Markov chain M_Δ where $\mathcal{C}(\Delta)$ is the set of states and the transitions are determined as follows:

- $p\varepsilon \xrightarrow{1} p\varepsilon$ for each $p \in Q$ (here ε denotes the empty stack);
- $pX\beta \xrightarrow{x} q\alpha\beta$ is a transition of M_Δ if and only if $pX \xrightarrow{x} q\alpha$ is a transition of Δ.

For every $pX\alpha \in \mathcal{C}(\Delta)$, where $X \in \Gamma$, a run initiated in $pX\alpha$ is *clean* if it does not reach a configuration $q\alpha$ for $q \in Q$. We denote $Clean(pX\alpha) \subseteq Run(pX\alpha)$ the set of all clean runs.

We define regular sets of configurations. Let $\mathcal{A}$ be a deterministic finite state automaton (DFA) with the alphabet $Q \cup \Gamma$. We put $\mathcal{C}(\mathcal{A}) := \{p\alpha \in \mathcal{C}(\Delta) \mid \text{reverse of } p\alpha \text{ is in } \mathcal{A}\}$ the set of configurations accepted by $\mathcal{A}$. A set $\mathcal{C} \subseteq \mathcal{C}(\Delta)$ is *regular* if there is a DFA $\mathcal{A}$ such that $\mathcal{C} = \mathcal{C}(\mathcal{A})$. Further, a set $\mathcal{C} \subseteq \mathcal{C}(\Delta)$ is called *simple* iff for every configuration $p\alpha$ the membership of $p\alpha$ to $\mathcal{C}$ depends only on the head of $p\alpha$.

3 The Results

In this section we justify the results presented in Table 1 and Table 2. Virtually, all model-checking problems for pPDA are undecidable provided no restriction is imposed on valuations. Therefore, we restrict our attention to *regular* valuations which assign regular sets of configurations to atomic propositions. Similarly, a *simple* valuation assigns a simple set to every atomic proposition.

3.1 Upper Bounds

A complete proof of the following theorem is presented in [5]. (However, no part of this theorem has been published elsewhere so far.)

Theorem 3.1 ([5]) *The model-checking problem for pPDA and the logics qPCTL and qPECTL* is in **EXPTIME** and **2-EXPTIME**, respectively. The program complexity of model-checking pPDA (or pBPA) and each of the logics qPCTL, qPCTL* and qPECTL* is in **EXPTIME** (or in **P**, resp.).*

The only missing result is a tight upper bound on model-checking pPDA and qPCTL* ([5] provides only **3-EXPTIME** upper bound which is not tight).

Theorem 3.2 *The model-checking problem for pPDA and qPCTL* is in **2-EXPTIME**.*

The rest of this subsection is devoted to the proof of Theorem 3.2. Let us fix a pPDA $\Delta = (Q, \Gamma, \rightarrow, Prob)$. Since $\mathcal{P}^{=0}(\varphi) \equiv \mathcal{P}^{=1}(\neg\varphi)$ we consider qPCTL* formulae containing only $\mathcal{P}^{=1}$. First, we consider formulae of the form $\mathcal{P}^{=1}\varphi$ where φ is an LTL formula (i.e., all state subformulae of φ are atomic propositions).

Lemma 3.3 *Let ν be a simple valuation. There is a DFA $\mathcal{A}$ such that $p\alpha \models^{\nu} \mathcal{P}^{=1}\varphi$ iff $p\alpha \in \mathcal{C}(\mathcal{A})$. Assuming that the set $\mathbf{S}_\Delta = \{pX \in Q \times \Gamma \mid \mathcal{P}(Clean(pX)) > 0\}$ is known in advance, the automaton $\mathcal{A}$ is computable in time $2^{|Q|\cdot 2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta|^{\mathcal{O}(1)}$.*

Proof (sketch). We begin with some notation adopted from [13]. Let us denote $Cl(\varphi)$ the set of all subformulae of φ. To every run w of Δ we assign its type $type(w) \subseteq Cl(\varphi)$ such that $\psi \in type(w)$ iff $w \models^{\nu} \psi$. Consider a run $w = v\,w'$ where v is a finite path and w' is a run with a type t'. Observe that t' and v uniquely determine the type of w. We denote $type(v, t')$ the type determined by v and t'.

Now we formally define the automaton $\mathcal{A}$. Let $\mathcal{A} = (K, \Sigma, \gamma, P_0, F)$ where

- $K = 2^{Q \times 2^{Cl(\varphi)}}$ is the set of states;
- $\Sigma = \Gamma \cup Q$ is the input alphabet;
- The transition function γ is defined as follows:
  - for all $X \in \Gamma$: $\gamma(P, X)$ consists of all $(p, t) \in Q \times 2^{Cl(\varphi)}$ such that *at least one one* of the following conditions holds
  - (i) runs of $Clean(pX)$ have the type t with nonzero probability
  - (ii) there is $(q, t') \in P$ and a path v from pX to $q\varepsilon$ such that $type(v, t') = t$

· for all $p \in Q$: $\gamma(P,p) = \emptyset$ if for all $(p,t) \in P$ we have $\varphi \in t$; and $\gamma(p,t) = Q \times 2^{Cl(\varphi)}$ otherwise (an arbitrary non-empty set would suffice).

- The initial state $P_0 \subseteq Q \times 2^{Cl(\varphi)}$ consists of all pairs (p,t) where $p \in Q$ and t is the unique type of the run $(p\varepsilon)^\omega$.
- The set of accepting states $F = \{\emptyset\}$.

We call a type t *probable* for $p\alpha \in \mathcal{C}(\Delta)$ if a run from $Run(p\alpha)$ has the type t with nonzero probability. Intuitively, the automaton reads the stack bottom up. After reading the reverse of a stack content $\alpha \in \Gamma^*$, the automaton $\mathcal{A}$ enters a state P where $(p,t) \in P$ iff t is probable for $p\alpha$. Finally, being in the state P (after reading reverse of α) and reading a control state $p \in Q$, the automaton $\mathcal{A}$ accepts iff $\varphi \in t$ for every type t probable for $p\alpha$. This means that φ is satisfied almost surely by a run initiated in $p\alpha$.

To compute γ we show that both conditions (i) and (ii) can be decided in time $2^{2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta|^{\mathcal{O}(1)}$. A deeper analysis of the LTL model-checking algorithm of [13] reveals that it decides whether an LTL formula is satisfied with a given probability on the condition that the stack is never erased. This is achieved by computing all types t such that a clean run has type t with nonzero probability. We need exactly this to decide the condition (i). The algorithm runs in time $2^{2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta|^{\mathcal{O}(1)}$. To decide the condition (ii), we have to decide whether for a given $(q,t') \in P$ there is a path v from pX to $q\varepsilon$ such that $type(v,t') = t$. This problem can be reduced to the reachability problem for non-probabilistic PDA using a procedure described in [13], which runs in time exponential in $|\varphi|$ and polynomial in $|\Delta|$. This reduction and the results of [11] give an algorithm for deciding (ii) running in time $2^{\mathcal{O}(|\varphi|)} \cdot |\Delta|^{\mathcal{O}(1)}$. Now the overall complexity of computing $\mathcal{A}$ is easy to compute. □

Till now we assumed valuations of atomic propositions to be simple. Using standard methods (see, e.g., [5]) one can extend the above results to regular valuations. Let us fix a regular valuation ν. Let $a_1, \ldots, a_k$ be all atomic propositions occurring in φ (we are still working with a formula $\mathcal{P}^{=1}\varphi$ where φ is an LTL formula) and let us assume that for each $1 \leq i \leq k$ there is a DFA $\mathcal{A}_i$ accepting $\nu(a_i)$. Denote K_1, $\ldots$, K_k the sets of states of $\mathcal{A}_1, \ldots, \mathcal{A}_k$, respectively. The following corollary can easily be proved using techniques presented in [5, Section 4.5].

Corollary 3.4 *There is a deterministic finite state automaton $\mathcal{A}$ such that $p\alpha \models^\nu \mathcal{P}^{=1}\varphi$ iff $p\alpha \in \mathcal{C}(\mathcal{A})$. Assuming that $\mathbf{S}_\Delta$ is known in advance, the automaton $\mathcal{A}$ is computable in time $(2^{|Q|\cdot 2^{\mathcal{O}(|\varphi|)}} \cdot |\Delta| \cdot |\varphi| \cdot \prod_{i=1}^k |K_i|)^{\mathcal{O}(1)}$ and has $2^{|Q|\cdot 2^{\mathcal{O}(|\varphi|)}} \cdot \prod_{i=1}^k |K_i|$ states.*

Proof of Theorem 3.2 First, we introduce some notation. To every qPCTL* state formula τ we assign sets $Sub(\tau)$ and $Ap(\tau)$ of all subformulae of the form $\mathcal{P}^{\sim\varrho}\psi$ and of all atomic propositions, respectively. To get correct complexity bounds we assume w.l.o.g. that each atomic proposition has at most one occurrence in τ.

In what follows, we assume that the set $\mathbf{S}_\Delta = \{pX \in Q \times \Gamma \mid \mathcal{P}(Clean(pX)) > 0\}$ has already been computed. The set $\mathbf{S}_\Delta$ can be computed in **PSPACE** (see, e.g., [5]), and thus computing $\mathbf{S}_\Delta$ does not affect the overall complexity.

Now, it suffices to prove the following claim: For any qPCTL* formula τ and a regular valuation ν there is a DFA $\mathcal{A}_\tau$ such that for all $p\alpha \in \mathcal{C}(\Delta)$ holds $p\alpha \in \mathcal{C}(\mathcal{A}_\tau)$ iff $p\alpha \models^\nu \tau$. The automaton $\mathcal{A}_\tau$ is computable in time $2^{|\Delta|\cdot 2^{\mathcal{O}(|\tau|)}} \cdot \left(\prod_{\psi\in Ap(\tau)} |\mathcal{A}_\psi|\right)^{\mathcal{O}(1)}$ and has $\prod_{\psi\in Sub(\tau)} 2^{|Q|\cdot 2^{\mathcal{O}(|\psi|)}} \cdot \prod_{\psi\in Ap(\tau)} |\mathcal{A}_\psi|$ states.

We proceed by induction on the structure of τ. For atomic propositions the statement is obvious. For boolean connectives it follows from closure properties of regular languages. Let $\tau \equiv \mathcal{P}^{=1}\varphi$ with $\psi_1, \ldots, \psi_k$ being maximal state subformulae of φ. We replace all occurrences of ψ_i in φ with an atomic proposition a_i, $1 \leq i \leq k$, and obtain an LTL formula ψ. Then we define a regular valuation ν' assigning $\mathcal{C}(\mathcal{A}_{\psi_i})$ to every a_i. Now, Corollary 3.4 gives us a DFA $\mathcal{A}$ where $p\alpha \in \mathcal{C}(\mathcal{A})$ iff $p\alpha \models^{\nu'} \mathcal{P}^{=1}\psi$ iff $p\alpha \models^\nu \mathcal{P}^{=1}\varphi$. The complexity estimates follow using arguments similar to those given in the discussion preceding Theorem 5.3.2 of [5]. □

3.2 Lower Bounds

It was proved in [4] that model-checking non-probabilistic PDA and CTL* is **2-EXPTIME**-complete. We show that this lower bound is valid also for pBPA and both qPCTL* and qPECTL*. Formally, we prove the following theorem.

Theorem 3.5 *The model-checking problem for pBPA and both qPCTL* and qPECTL* is* ***2-EXPTIME****-hard.*

Proof (sketch). We proceed by reduction from the **2-EXPTIME**-complete acceptance problem for alternating exponentially space bounded Turing machines [7]. An alternating exponentially bounded Turing machine is a tuple $\mathcal{M} = (Q, K, J, q_0, \vdash, \dashv, \#, \delta, P)$ where Q is a finite set of control states, K is a finite input alphabet, $J \supseteq K$ is a finite tape alphabet, $q_0 \in Q$ is the initial state, $\vdash, \dashv \in J$ are markers of the left-hand and right-hand side of a tape, $\# \in J$ is an empty symbol, $\delta : Q \times J \to 2^{Q\times J\times\{L,R\}}$ is a transition function, and $P = (Q_\forall, Q_\exists, Q_{acc}, Q_{rej})$ is a partition of Q into universal, existential, accepting and rejecting states, respectively. W.l.o.g. we assume that $Q \cap J = \emptyset$, $q_0 \in Q_\exists$ and $\delta(q, a)$ has exactly two elements (q_1, a_1, D_1), (q_2, a_2, D_2) for all q, a, where $D_1 \neq D_2$ and $q_1 \neq q_2$. A configuration of $\mathcal{M}$ consists of a control state together with a position of the head on the tape and a word $w \in J^*$ such that $|w| = 2^{rj}$ where r is a constant, $j > 1$ is the size of the input word and $w(0) = \vdash$ and $w(2^{rj} - 1) = \dashv$. We write configurations as words $vqv' \in (Q \cup J)^*$ where vv' is the content of the tape, q is a control state and $|v|$ is the position of the head (i.e., the machine reads the first letter of v'). A configuration vqv' is accepting if $q \in Q_{acc}$.

A *computation tree* on an input word $v \in K^*$ is a tree T satisfying the following: The root of T is labeled by the initial configuration $q_0{\vdash}v\#^i{\dashv}$ where $i = 2^{r|v|} - |v| - 2$, and if N is a node labeled by a configuration with a control state q, then the following holds:

- if q is accepting or rejecting, then N is a leaf;
- if q is existential, then N has one successor labeled by a configuration reachable from the configuration of N in one step;

- if q is universal, then N has two successors which are labeled by both configurations reachable from the configuration of N in one step.

We can safely assume that all computation trees of $\mathcal{M}$ are finite. A computation tree is accepting if all its leaves are labeled by accepting configurations. The machine $\mathcal{M}$ accepts v if there is an accepting computation tree on the input word v.

For a given $\mathcal{M}$ and an input word v we construct a pBPA $\Delta = (\Gamma, \rightarrow, Prob)$, a configuration $\sigma \in \Gamma^*$ and a formula Φ, such that $\sigma \models^{\nu} \Phi$ iff v is accepted by $\mathcal{M}$. We use the symbols from Γ as atomic propositions and define ν to be the simple valuation assigning the set $\{X\}\Gamma^*$ to each proposition X. We put $n = r|v|$. Let us denote $m = 2^n - 2$.

We use the fact that one can effectively compute a set of *compatible* hex-tuples $Comp_L(\mathcal{M}) \subseteq (J \cup Q)^6$ such that for all configurations ω and ω' of $\mathcal{M}$ we have that ω' is reachable from ω in one step while moving head to left iff $(\omega(i), \omega(i+1), \omega(i+2), \omega'(i), \omega'(i+1), \omega'(i+2)) \in Comp_L(\mathcal{M})$ for all $0 \leq i \leq m$. Similarly, we define $Comp_R(\mathcal{M})$ for moving head to right and $Comp(\mathcal{M}) = Comp_L(\mathcal{M}) \cup Comp_R(\mathcal{M})$.

Intuitively, the pBPA Δ simulates the depth first search on a computation tree by guessing configurations on its stack as sequences of triples. Each configuration ω is represented by a sequence $[y_m], [y_{m-1}], \ldots, [y_0]$ where $y_i = \omega(i)\omega(i+1)\omega(i+2)$ for $0 \leq i \leq m$. Whenever a new configuration is guessed, the pBPA Δ checks whether two triples on the same position in two adjacent configurations form a compatible hex-tuple. To deal with exponentially many triples in each configuration, we adopt the construction of [4]. The idea is based on encoding the position of each triple using n binary counters.

Formally, the set Γ consists of all $[y]$ where $y \in (J \cup Q)^3$ and distinguished symbols G, F, A, L, R, E, T, H, 0 and 1. The transition relation $\rightarrow$ is defined as follows:

$$
\begin{array}{lll}
H \rightarrow H[\#^3]c \mid G[y]0^n E & G \rightarrow G[y]0^n L \mid G[y]0^n E \mid G[y]c \mid A \mid F & L \rightarrow G[y]0^n R \mid F \\
[y] \rightarrow \varepsilon \mid T & R \rightarrow \varepsilon \mid F & E \rightarrow \varepsilon \mid F \\
0 \rightarrow \varepsilon & 1 \rightarrow \varepsilon & T \rightarrow \varepsilon \\
F \rightarrow \varepsilon & A \rightarrow \varepsilon &
\end{array}
$$

Here, c denotes a word from $\{0,1\}^n$. Note that the rules $H \rightarrow H[\#^3]c$ and $G \rightarrow G[y]c$ are in fact abbreviations for families of rules that guess c on the top of the stack symbol by symbol and then add $H[\#^3]$ and $G[y]$, respectively, on the top. Also, each rule with the right-hand side longer than two symbols can be decomposed into several rules with right-hand sides of length at most two.

In the following text, we use $c_k \in \{0,1\}^n$ to denote the binary representation of the number k. Let $[y_m], [y_{m-1}], \ldots, [y_0]$ be the sequence of triples encoding the initial configuration of $\mathcal{M}$. By $PConf$ we denote the set of all finite words of the form $[y]c[y']c' \ldots$, i.e., sequences of triples followed by numbers. Let us consider a run w of Δ initiated in $\sigma = H[y_n]c_n \ldots [y_0]c_0$ satisfying the following properties:

(i) w reaches ε and never enters F;

(ii) each configuration of w with the head $U \in \{G, H\}$ is either of the form $U[y]c_k[y']c_{k-1}\gamma$, where $k \leq m$, or $U[y]c_0X[y']c_m\gamma$ where $X \in \{L, R, E\}$;

(iii) each configuration of w starting with G has the form $G[y]c_k\alpha X\beta[y']c_k\gamma$ where $0 \leq k \leq m$, $X \in \{L, R, E\}$, $\alpha, \beta \in PConf$, and (y, y') is from $Comp_L(\mathcal{M})$, $Comp_R(\mathcal{M})$ or $Comp(\mathcal{M})$, depending on whether $X = L$, $X = R$ or $X = E$;

(iv) each configuration of w with the head A can be written as $A\alpha[y]\beta$ where $\alpha \in PConf$ and y contains $q \in Q_{acc}$.

It is easy to prove that w uniquely corresponds to a depth-first search on some accepting computation tree of $\mathcal{M}$. Observe that Δ is forced, by the property (ii), to first reach $H[y_m]c_m \dots [y_0]c_0$, and then to simulate an accepting computation of $\mathcal{M}$ from the initial configuration encoded by $[y_m], \dots, [y_0]$. We define the formula Φ as follows

$$\Phi \equiv \mathcal{P}^{>0}\Big(\Diamond\varepsilon \wedge \neg\Diamond F \wedge \Box\big(((G \vee H) \Rightarrow Counter) \wedge (G \Rightarrow Check) \wedge (A \Rightarrow Acc)\big)\Big)$$

Intuitively, Φ should assure that there is a run satisfying conditions (i)–(iv). It can be easily formulated in qPECTL*, provided that $Counter$, $Check$ and Acc are qPECTL* formulae. In the rest of this proof we define the formulae $Counter$, $Check$, and Acc to assure the conditions (ii), (iii), and (iv), respectively.

Let us start with $Check$. Although the condition (iii) can be expressed straightforwardly using qPCTL* formula, the logic qPECTL* demands more careful approach. In order to avoid an exponential blow-up in the size of the formula we express *negation* of the condition (iii) and consequently negate the formula.

Let $G\gamma$ be a configuration which does not satisfy (iii) and is reachable from $H[y_n]c_n \dots [y_0]c_0$. All runs w initiated in $G\gamma$ can be classified as follows (here $h[w] = head(w(0))\, head(w(1)) \dots$ is the string of heads of configurations of w):

- $h[w](1) \neq F$, or $h[w]$ starts with $GF[y]T$;
- $h[w]$ starts with $GF[y]c\alpha Xz$ where $X \in \{L, R, E\}$, $\alpha \in (\Gamma\backslash\{L, R, E\})^*$, and $z \neq F$;
- $h[w]$ starts with $GF[y]c\alpha X\alpha' X'$ where $X, X' \in \{L, R, E\}$, $\alpha \in (\Gamma\backslash\{L, R, E\})^*$, and $\alpha' \in (\Gamma\backslash\{T, L, R, E\})^*$;
- $h[w]$ starts with $GF[y]c\alpha XF\alpha'[y']Tc'$ where $X \in \{L, R, E\}$, $\alpha \in (\Gamma\backslash\{L, R, E\})^*$, $\alpha' \in (\Gamma\backslash\{T, L, R, E\})^*$, and one of the following holds:
  - $c \neq c'$;
  - $([y], [y'])$ is not from $Comp_L(\mathcal{M})$, $Comp_R(\mathcal{M})$ or $Comp(\mathcal{M})$ depending on whether $X = L$, $X = R$ or $X = E$.

Note that for each of the above cases the set of all words $h[w]$ is accepted by a non-deterministic Büchi automaton computable in polynomial time. Now we define $Check = \neg\mathcal{P}^{=1}(\mathcal{B}_{Check}(Y_1, \dots, Y_k))$ where $\Gamma = \{Y_1, \dots, Y_k\}$ and the automaton $\mathcal{B}_{Check}$ nondeterministically guesses which one of the above cases occurs and then verifies it (note that always precisely one of $Y_1, \dots, Y_k$ is satisfied, and hence we may assume that transitions of $\mathcal{B}_{Check}$ are labeled with letters of Γ).

Now we define $Counter$. Let $U \in \{G, H\}$ and let $U\gamma$ be a configuration which does not satisfy (ii). All runs w from $U\gamma$ can be classified as follows (here $X \in \{L, R, E\}$):

- $h[w](1) \neq F$, or $h[w]$ starts either with $UF[y]T$, or $UF[y]c[y']T$, or $UF[y]cXF[y']T$;
- $h[w]$ starts with $UF[y]c_kXF[y']c_m$ where $k \neq 0$;
- $h[w]$ starts with $UF[y]c_k[y']c_\ell$ where $\ell = m$ or $k \neq \ell + 1$.

Clearly, a Büchi automaton $\mathcal{B}_{Counter}$ accepting the set of all words $h[w]$ satisfying the above conditions is computable in polynomial time. We define $Counter = \neg\mathcal{P}^{=1}(\mathcal{B}_{Counter}(Y_1, \ldots, Y_k))$.

Finally, let us define $Acc = \mathcal{P}^{>0}(\mathcal{B}_{Acc}(Y_1, \ldots, Y_k))$ where the automaton $\mathcal{B}_{Acc}$ accepts all words of the form $A\alpha[y]\beta$ where $\alpha \in PConf$ and $[y]$ contains $q \in Q_{acc}$. The automaton $\mathcal{B}_{Acc}$ can easily be constructed in polynomial time.

It is straightforward to verify that formulae $Counter$, $Check$ and Acc can also be defined in qPCTL*. □

References

[1] P.A. Abdulla, C. Baier, S.P. Iyer, and B. Jonsson. Reasoning about probabilistic channel systems. In *Proceedings of CONCUR 2000*, vol. 1877 of *LNCS*, pp. 320–330. Springer, 2000.

[2] C. Baier. *On the Algorithmic Verification of Probabilistic Systems*. Habilitation, Universität Mannheim, 1998.

[3] C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems: an algorithmic approach. In *Proceedings of 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, vol. 1601 of *LNCS*, pp. 34–52. Springer, 1999.

[4] Laura Bozzelli. Complexity results on branching-time pushdown model checking. In E. Allen Emerson and Kedar S. Namjoshi, editors, *Proceedings of VMCAI*, vol. 3855 of *Lecture Notes in Computer Science*, pp. 65–79. Springer, 2006.

[5] T. Brázdil. *Verification of Probabilistic Recursive Sequential Programs*. PhD thesis, Faculty of Informatics, Masaryk University, 2007.

[6] T. Brázdil, A. Kučera, and O. Stražovský. On the decidability of temporal properties of probabilistic pushdown automata. In *Proceedings of STACS'2005*, vol. 3404 of *LNCS*, pp. 145–157. Springer, 2005.

[7] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[8] M. Dam. Ctl* and ectl* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[9] L. de Alfaro. Temporal logics for the specification of performance and reliability. In *Proceedings of STACS'97*, vol. 1200 of *LNCS*, pp. 165–176. Springer, 1997.

[10] J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. In *Proceedings of LICS 2004*, pp. 12–21. IEEE, 2004.

[11] J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. *I&C*, 186(2):355–376, 2003.

[12] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic systems. In *Proceedings of TACAS 2005*, vol. 3440 of *LNCS*, pp. 253–270. Springer, 2005.

[13] K. Etessami and M. Yannakakis. Checking LTL properties of recursive Markov chains. In *Proceedings of 2nd Int. Conf. on Quantitative Evaluation of Systems (QEST'05)*, pp. 155–165. IEEE, 2005.

[14] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. In *Proceedings of STACS'2005*, vol. 3404 of *LNCS*, pp. 340–352. Springer, 2005.

[15] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.

[16] J. G. Kemeny and J. L. Snell. *Finite Markov Chains: With a New Appendix "Generalization of a Fundamental Matrix"*. Springer, first edition, 1983.

[17] J. G. Kemeny, J. L. Snell, A. W. Knapp, and D.S. Griffeath. *Denumerable Markov Chains*. Springer, second edition, 1976.